Estructura de Datos y Algoritmos

Grados en Ingeniería Informática, de Computadores y del Software

Examen Parcial de Junio - 1 de junio de 2018 - Grupos C y F

Nombre: _____ **Grupo:** _____

Laboratorio: _____ **Puesto:** _____ **Usuario de DOMjudge:** _____

1. (4 puntos) Un grupo de emprendedores madrileños ha decidido formar una empresa dedicada a la impartición de másteres y cursos online. Para ello es crucial disponer de una aplicación que permita gestionar la impartición de este tipo de cursos. El primer prototipo simplemente ha de gestionar un solo curso/máster online incluyendo las siguientes funcionalidades: añadir estudiantes al curso, añadir tareas al curso así como cambiar su peso en la nota final, poner notas, calcular notas finales, y obtener diferentes listados. En particular, el TAD `CursoVirtual` debe contar con las siguientes operaciones:

- `CursoVirtual()`: Crea un curso vacío.
- `addStudent(SId)`: Añade al curso un nuevo estudiante identificado unívocamente por `SId` (tipo `string`). Si ya existe un estudiante con el mismo identificador se produce un error.
- `addAssignment(A,W)`: Añade una nueva tarea de nombre `A` (tipo `string`), usado como identificador unívoco, y peso `W` (tipo `double` en el rango 0..1). Si ya existe una tarea con el mismo nombre se produce un error.
- `setMark(SId,A,M)`: Pone la nota `M` a la tarea `A` del estudiante `SId`. La nota es un valor numérico (tipo `double`) en el rango 0..10. Si ya había una nota puesta para dicha tarea se sobrescribe por la nueva proporcionada. Si el estudiante o la tarea no existen se produce un error.
- `getFinalMark(SId,M)`: Calcula y devuelve la nota final `M` del estudiante `SId`. La nota se calcula sumando las notas obtenidas por el estudiante en todas las tareas, cada una multiplicada por su correspondiente peso (las tareas no realizadas cuentan 0). Si el estudiante no existe se produce un error.
- `setAssignmentWeight(A,W)`: Sobrescribe con `W` el peso de la tarea `A`. Si la tarea no existe se produce error.
- `listOfStudents(L)`: Obtiene en `L` la lista de estudiantes ordenada por su identificador.
- `export(FN)`: Escribe en el fichero `FN` las notas de todos los estudiantes en forma de tabla.

Se pide:

a) **[0.75 puntos]** Elegir una representación adecuada para la implementación del TAD `CursoVirtual` utilizando los TADs proporcionados, de forma que la complejidad de la operación `listOfStudents` sea como máximo $\mathcal{O}(n)$, siendo n el número de estudiantes del curso, y, la de `setAssignmentWeight` sea $\mathcal{O}(1)$.

b) **[0.75 puntos]** Indicar y justificar brevemente el coste asintótico esperado de todas las operaciones del TAD con la representación elegida.

c) **[1.5 puntos]** Implementar las operaciones `setMark`, `getFinalMark` y `listOfStudents`. Se valorará la complejidad en tiempo y espacio de las soluciones obtenidas.

d) **[1 punto]** Se desea añadir la operación `getEmpollones` al TAD, que devuelve un listado con los estudiantes que han realizado todas las tareas, ordenado por identificador.

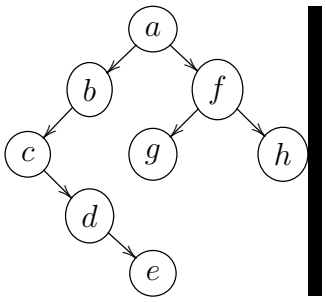
d1) ¿Qué coste tendría si utilizamos la representación del apartado a)?

d2) Mejora la representación del TAD de manera que la nueva operación sea lo más eficiente posible sin empeorar (o empeorando lo menos posible) el resto de operaciones. Indica el coste de la nueva operación y el de las operaciones que hayan cambiado.

2. (2 puntos) Implementa una función (externa al TAD List) que dada una lista enlazada de caracteres (`List<char>`) `l`, y dos números enteros `i` y `k`, invierta el segmento que comienza en la posición `i`-ésima de la lista `l` (se empieza desde la posición 1) y contiene `k` elementos. Por ejemplo, si la lista está formada por los elementos *a b c d e f*, e invertimos el segmento que comienza en la posición 3 y tiene longitud 3, entonces la lista resultante es *a b e d c f*. Si $i + k > l.size() + 1$ entonces se invertirá el segmento que va desde la posición `i`-ésima hasta el final de la lista. Se valorará la complejidad en tiempo y en espacio de la función implementada, la cual debes indicar y justificar.

3. (4 puntos) En un árbol binario, se define el *factor de equilibrio* de un nodo como el valor absoluto de la diferencia entre las alturas de sus dos hijos. Decimos que un nodo es *equilibrado* si su factor de equilibrio es 0 o 1. Por otro lado decimos que un nodo es *padre* si al menos tiene un hijo (no vacío), y es un *buen padre*, si es padre y sus hijos son equilibrados. Se pide:

a) **(2 puntos)** Implementar una función que dado un árbol binario de caracteres devuelva el porcentaje (como entero redondeado hacia abajo) de nodos padres que son buenos padres. En el árbol del ejemplo hay 5 padres y 3 de ellos (c, d y f) son buenos padres. Se debe devolver por tanto 60. Aclaración: dados dos enteros n y m ($n < m$) debes calcular el porcentaje mediante `int(double(n)*100/m)`. Si `m` = 0 debes devolver 100.



b) **(2 puntos)** Implementar un programa que imprima un listado de todos los nodos del árbol que no son equilibrados, incluyendo para cada uno, el factor de equilibrio, el camino para llegar a él desde la raíz, y el valor del nodo. El listado debe ir ordenado de menor a mayor factor de equilibrio, y en caso de empate, por orden lexicográfico del camino. Para ello debes: (1) definir el tipo `TablaDesequilibrados` para almacenar el listado; (2) implementar una función que dado un árbol binario de caracteres devuelva una `TablaDesequilibrados`; y (3) implementar una función que dada una `TablaDesequilibrados` imprima el listado (ver plantilla y casos de prueba proporcionados). Aclaración: Si utilizas un `string` de 0's y 1's para representar el camino (0 significa ir por el hijo izquierdo y 1 ir por el derecho) el operador de orden del tipo `string` implementa precisamente el orden lexicográfico. En el ejemplo, los nodos desequilibrados son: a y c, con factor 2 y caminos "" y "00" resp., y b con factor 3 y camino "0".

Instrucciones para la realización y entrega del examen

- Las plantillas y casos de prueba para poder probar vuestras soluciones, así como los TADs del 2º cuatr., se obtienen pulsando en el icono del Escritorio "Publicacion docente ...", después en "Alumno recogida docente", y en el programa que se abre, abriendo en la parte derecha la carpeta EDA-CF, arrastrando los ficheros a hlocal (en la izqda).
- Al principio del fichero principal de cada ejercicio debe aparecer, en un comentario, tu nombre completo, dni y puesto de laboratorio. También debes incluir unas líneas explicando lo qué has conseguido hacer.
- Todo lo que no sea código C++ (explicaciones, costes, etc.) debe ir en los propios ficheros en comentarios debidamente indicados.
- La entrega se realiza por el juez online accesible en la dirección `http://exacrc/domjudge/team`. En el juez te identificarás con el nombre de usuario y contraseña que has recibido al comienzo del examen. Tus soluciones serán evaluadas independientemente del veredicto del juez. Se tendrá en cuenta **exclusivamente** el último envío de cada ejercicio.